一种 B-树和 Bloomfilter 相结合的 IPv6 路由查找算法 *

姚　明，赵晶晶，贺兴亚，杨　云

(扬州大学 信息工程学院，江苏 扬州 225005)

摘　要： 为了提高 IPv6 的路由查找效率，针对 IPv6 路由前缀分布不均匀的问题，提出了一种基于 B-树和 Bloom filter 相结合的 IPv6 路由查找算法（BTBF）。BTBF 分为 B-树和 Bloom filter 查找两部分，首先利用 B-树查找路由前缀的前 16 bit 值，然后通过 B-树节点中位向量的映射，将下一步链接到 Bloom filter，再利用 Bloom filter 位数组的值映射提取下一跳。实验结果表明，BTBF 算法与其他树形和 Bloom filter 类算法相比有效减少了空间和时间占用，在路由表项数变化较大的情况下也能维持稳定的查找性能。

关键词： 路由查找算法；B-树；Bloom filter；IPv6 骨干路由表

中图分类号： TP393.03　　doi: 10.3969/j.issn.1001-3695.2018.03.0179

Fast IPv6 routing lookup combining B-tree and Bloom filter

Yao Ming1, Zhao Jing Jing1, He Xing Ya1, Yang Yun1

(1. *Information Engineering College of YangZhou University*, *Yangzhou* 225002, *China*)

Abstract: In order to improve the efficiency of IPv6 routing lookup, this paper put forward an IPv6 routing lookup algorithm (BTBF) based on the combination of B-Tree and Bloom filter for the uneven distribution of IPv6 routing prefix. BTBF includes two parts, namely, B-Tree and Bloom Filter lookup: firstly, this paper looked up the first 16bit value of routing prefix with B-tree, and linked the next step to Bloom filter through the mapping of middle bit vector of B-tree node; then, extracted the Next hop through the value mapping of Bloom filter bit array. The experimental results show that, compared with other Tree and Bloom filter algorithms, BTBF algorithm can effectively reduce space and time, and maintain stable routing performance under the condition of larger changes in Routing-Table Entries.

Key words: routing lookup algorithm; B-tree; Bloom Filter; IPv6 backbone routing table

0　引言

IPv6 得到了产业界和学术界的广泛关注和认可，中国的下一代互联网 CNGI（China next generation Internet）与美国联邦机构也早已对 IPv6 进行了预先部署[1]。IPv6 地址长度从 IPv4 的 32bit 变为 128bit，使得互联网的骨干网路由器在进行表项查找时遇到巨大的性能瓶颈。因此，必须研究针对 IPv6 路由表的可靠快速的查找机制[2][3]。

IP 路由查找算法可分为基于前缀长度和基于前缀值的查找算法。基于前缀长度的算法多采用树形数据结构，不过在推广至 IPv6 查找时，IP 地址长度增长至 IPv4 的 4 倍，树形结构的查找深度变大从而降低了查找效率。基于前缀值的查找算法可以避免查找树的较大深度的问题，Zhong[4]提出了一种以 B-树为基础的递归平衡多路径区间树（RBMRTs）的数据结构进行最长前缀匹配，查找和更新的时间复杂度均为 O($\log_m$N)。但是，RBMRTs 的内部节点中存储了许多重复点，路由更新时增加了时间和内存需求。BSRPS 算法[5]将相邻的前缀值区间作为一个集合放置在一个节点中以减少查找树产生多余的空置与重复节点，但是该算法面对 IPv6 的海量地址空间时依然产生很多空置节点，大大增加了访存次数。Dharmapurikar 等人提出了基于 Bloom filter 的最长前缀匹配算法[6]，该算法将不同长度的 IP 地址前缀，分别存储在不同的 Bloom filter 中，在进行查找时，将待查找 IP 地址并行输入到各个过滤器中进行查询。该算法容易产生过多的误判，而且路由表中不同前缀长度对应的路由表的规模大小差异很大，无法保证找到一个适应力强的 Bloom filter。基于 TCAM 的高效浮动关键词匹配算法[7]中采取前缀扩展的方式减少了 Bloom filter 的数量，但是同时也扩大了路由表，降低了查找速度。基于 Bloom 滤波器的快速路由查找方法[8] 主要在索引表中探测首字节以减少 Bloom filter 查询次数，但是未对首字节长度做出规定，因此首字节过滤效率不明确，而且在 IPv6

收稿日期： 2018-03-18；**修回日期：** 2018-05-02　　**基金项目：** 江苏省产学研前瞻性资助项目（BY2016069）

作者简介： 姚明（1993-），男，陕西西安人，硕士研究生，主要研究方向为网络信息与安全；赵晶晶（1993-），女，江苏扬州人，硕士研究生，主要研究方向为无线传感器网络；贺兴亚（1974-），女，江苏扬州人，讲师，硕士，主要研究方向为 QoS 路由算法；杨云（1967-），男，江苏扬州人，教授，硕导，博士，主要研究方向为 TCP/IP 协议分析、无线网络 QoS 路由算法.

海量地址空间基础上索引表会增加查找时间。

针对以上问题，在当前骨干路由器 IPv6 路由表数据统计分析的基础上，结合 B-树和 Bloom filter，提出了一种 IPv6 路由查找算法。因为 B-树在查询数量较多的路由表项时会出现大量重复节点的现象，所以仅用 B-树查找前 16bit 前缀值，并形成对 Bloom filter 的索引，然后对路由表进行进一步的查找。B-树深度小，能有效避免空置集合，在 B-树的索引下，降低 Bloom filter 需要查找的路由表项规模、误判率和查找时间。

1　IPv6 地址结构和骨干路由表特点

1.1　IPv6 地址结构

RFC3587 规定的最新 IPv6 全球单播地址结构如图 1 所示，由 3 个部分组成：n bit 的全球路由前缀、64-n bit 的子网标志和 64 bit 的接口标志。全球路由前缀标志一个站点，子网标志用于标志站点内的一个子网，接口标志是用 IEEE EUI-64 格式生成的用来表示子网内的主机，用于路由的前缀长度最长为 64 bit。根据对骨干路由器的统计分析得知 IPv6 前缀长度最小为 19 bit。

n bit	64-n bit	64bit
全球路由前缀	子网 ID	接口 ID

图 1　IPv6 全球单播地址结构

1.2　骨干路由表特点

对骨干路由器 potaroo[9]自治域 AS131072 的所有 8 万余条路由表项的统计分析，可以发现目前骨干路由器中 IPv6 地址有如下特点：

a）如图 2 所示，对前 16 bit 取值统计发现，无前缀长度小于 16 bit 的路由表项，前 16 bit 取值有 39 个，其分布不均匀且数目差别较大。基于前缀值的查找方法不能有效匹配到数量较多的前缀集合中，在针对 IPv6 进行路由查找时，一方面需要增加路由查找取值较多集合的概率，另一方面需要减少路由前缀值查找树的深度，以便更快的找到所需要探测的集合，BTBF 算法中采用 B-树以实现更低的查找树深度。

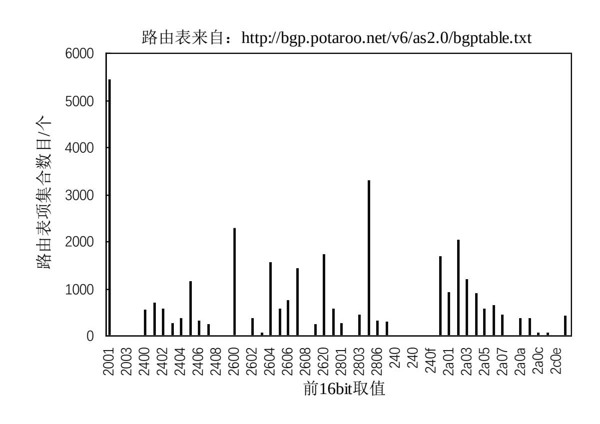


图 2 potaroo 路由器所有前 16bits 取值统计

b）potaroo 官网数据显示，IPv6 路由表由前缀长度值从/16 到/64 的分布差异很大，对 AS131072 路由表进一步统计分析后得到图 3，图中一条线代表了一个前 16 bit 取值的集合、在不同前缀长度的路由表前缀集合数目变化。

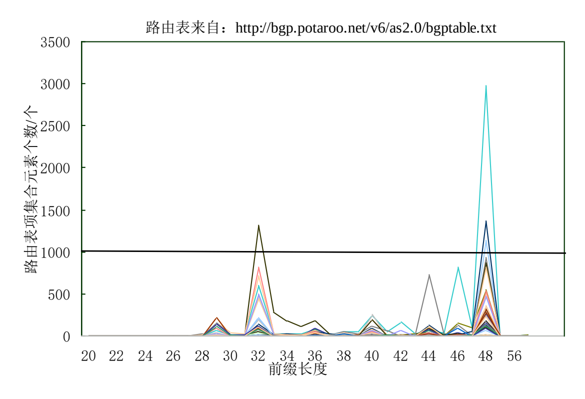


图 3 potaroo 路由表项前 16bits 取值及长度统计

虽然前 16bit 前缀值形成的各集合元素个数差异较大，但是从图 3 中可以看出，其分布的特征基本相同。利用 B-树和前缀长度值，把整个路由表项划分成很多个元素个数较少的集合，图中每一个峰值对应一个划分好的集合，代表 Bloom Filter 需要查找的路由表项。为了简化算法设计，对所有集合的数目设定一个阈值用来划分级别，图 3 中的虚线位置 1000 将这些集合划分了千级和百级。这样 Bloom Filter 可以针对划分好的级别，得到最优的 Hash 函数个数和位数组大小，以便在降低错误率的同时提升查找效率。

2　基于 B-树和 Bloom filter 的路由查找算法

IPv6 路由表中前 16 bit 取值较少，在 2001、2600 和 2804 上较为集中，其余的取值路由表项较少。BTBF 首先利用 B-树对前 16bit 的取值进行查询，将每个对应的路由表依据前缀长度进行分类，如 potaroo 骨干路由器可以从/19-/64 共分为 46 个组。每组对应一个 Bloom filter，对不同规模的组，采取不同数量的 hash 函数和位数组大小，降低无效探测的次数。

2.1　数据结构

2.1.1 前 16 bit 值的 B-tree 树型结构

B-tree 的查找、插入和删除操作皆以“对数时间”的速度完成，它是一种自平衡的树，所以 B 树不需要像其他平衡查找树那样频繁地重新保持平衡。子节点数量的上界和下界，根据特定的实现而设置，根据前 16 bit 取值出现的数量较少，数值分布不均匀的特征，设定 B 树子节点数量的上界为 2，下界为 3，通常这种树也被称之为 2-3 树。该树有三个重要特征:非空节点上的“数据项”大于等于 1 且小于等于 2；每个节点最多有 3 个子节点；该树是自平衡树。在进行更新时需要将 4-node 转换为 3-node 或 2-node，这个转换操作只需要在常数量级的时间内即可实现。

2.1.2 B-tree 节点的数据结构

B-tree 的节点类型分为叶子节点和非叶子节点，每个节点上存储有 1-2 个数据项，这些数据项存储了前 16 位前缀值。节点与 Bloom filter 的转发操作利用到两个长 m bit 的位向量：$V_S[h(/19), h(/20), h(/21),\ldots, h(/64)]$和 $V_L[h(/19), h(/20), h(/21),\ldots,$

h(/64)]。V_S 和 V_L 分别存储了集合元素的映射信息，向量内的 Hash 函数 *h*(/*n*)会根据前缀长度 *n* 将集合映射到向量 *V* 中，其中树节点的数据结构如图 4 所示。

S	*L*	*CNUMBER*	
LNode	*MNode*	*RNode*	
$V_S[h(/19),h(/20),h(/21),\cdots,h(/64)]$			
$V_L[h(/19),h(/20),h(/21),\cdots,h(/64)]$			

图 4 B-tree 节点数据结构

该节点含有两个数据项 *S* 和 *L*，标志位 *CNUMBER*，三个指针 *LNode*，*MNode* 和 *RNode*，及两个向量 V_S 和 V_L。*S* 和 *L* 分别为节点中较小的前缀值和较大的前缀值，各 16 位。*CNUMBER* 为该节点的子节点数量，标记节点类型为 2-node、3-node 或叶子节点，共有 2 位，取值可为 11、10 或 00。如果 *CNUMBER* 为 11 该节点则为 3-node，并表示节点最多有 3 个子节点，节点中存储全部的数据项、子节点指针和向量；如果 *CNUMBER* 为 10 该节点则为 2-node，并表示节点最多拥有两个子节点，数据项仅保留 *S*，向量仅保留 V_S；如果 *CNUMBER* 的值为 00 时表示当前节点无子节点即为叶子节点。*LNode*，*MNode* 和 *RNode* 分别为节点的左中右子节点指针。V_S 和 V_L 为节点中的两个向量，映射路由前缀前 16bit 的值为 *S* 或 *L* 的路由表项集合，该向量的每个元素为 hash 值 *h*(/*n*)，*h*()为 hash 函数，*n* 为前缀长度。*h*(/*n*)表示前缀长度为 *n* 的路由表项集合地址。根据目前 IANA 的 IPv6 地址分配策略，骨干路由器中无前缀长度小于 19 的路由表项，因此向量的元素从 *h*(/19)至 *h*(/64)有序存储。路由查找两个阶段的转发操作利用到了 V_S 和 V_L：当 B-树节点匹配成功时，算法选取数据项 *S* 或者 *L* 对应的向量，读取向量元素存储的路由集合地址，转发至 Bloom filter 查询。

2.1.3 Bloom filter 的数据结构

如图 5 所示，首先，Bloom filter 初始化后有一个 *m* 位的位数组，数组元素全为 0。同时，定义了 *k* 个不同的 Hash 函数，每一个 hash 函数都随机的将输入元素映射到位数组中的一个位上。那么在没有发生误判的情况下，对于一个确定的输入，都会得到一个确定的位数组值。

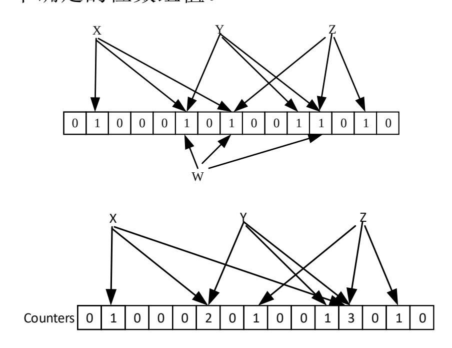


图 5 基本 Bloom filter 与改进后的 Bloom filter

在标准 Bloom filter 算法中常常会碰到哈希算法中的冲突（碰撞）问题，不同数值经过哈希得到的两个结果值有可能相同，这时就会发生误判[10]。图 5 中，当插入 x、y、z 这三个元素之后，再来查询 w 是否在集合之中，而如果 w 经过三个 hash 函数计算得出的结果所得索引处的比特位全是 1，那么 Bloom filter 则得出 w 在集合之中，实际上这里是误报，w 并不在集合之中。标准 Bloom filter 不能进行删除操作，改用 counters 计数数组改进原有的位数组。在插入元素时给计数数组的 *k* 个元素的值分别加 1，删除元素时给计数数组 *k* 个元素的值分别减 1。

Bloom filter 用来探测 B-树的匹配节点转发到的路由表项集合，具体方案是：首先，读取 *V*[*h*(/19), *h*(/20), *h*(/21),…,*h*(/*n*)]向量各个非空元素值，得到 *m* 个集合的地址（$1<m<n-18$）,每一个集合都对应了一个 Bloom filter，所以共有 *m* 个 Bloom filter。然后，将数据包的目的 IP 地址并行输入到这 *m* 个 Bloom filter 中查询。最后，根据匹配到的 Bloom Filter 计数数组的取值索引路由表，提取下一跳地址。

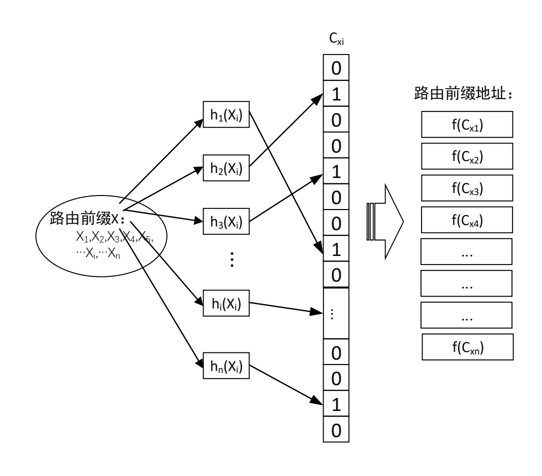


图 6 路由前缀集合存储与地址映射

在最后一个步骤中，Bloom filter 仅能探测有无符合目标 IP 地址的最长前缀，它的本身是不存储路由表的，具体的下一跳地址需要进一步地搜索路由表。即使骨干路由表按照前缀长度进行了分类，但是某些长度的集合表项依然十分庞大，传统的搜索提取会对查找性能造成影响，所以提出 Bloom Filter 的映射方式对目标路由表项进行快速提取。Bloom Filter 中插入一个新的前缀 X 时，得到 counters 位数组的值 C_n。如图 5，X 的值 C_x = 0100010000 01000 = 0x2208。通过 C_n 的值建立地址映射 f_x (C_n)，此地址映射就是目标 IP 地址前缀在路由表中的具体位置。通过这种方法可以对 IP 数据包进行快速查找并转发。图 6 给出了路由表项的存储及各表项地址的关系示意图。

2.2 路由查找

路由查找的伪代码如下所示：

```
input:dstIP; //目的 IP 地址
output:next hop;
search BTBF(dstIP){
BMP = next hop of default route;
key = the 1-16bit of dstIP;
node = search B-Tree of order 3(key);
if(node.CNUMBER == 2) node.S = node.L;
if(key !=node.S || key != node.L){
```

```
7.  if(key < node.S ) node = node.LNode;
8.  if(key > node.S && key < node.L) node = node.MNode;
9.  if(key >node.L) node = node.RNode;
10. search BTBF(dstIP);
11. }else{
12. if(node.CNUMBER == 0) return BMP;
13. if(key = node.S return Vs[];
14. if(key = node.L return VL[];
15. }
16. i = the length of route prefix;
17. counters = BinaryArray of Bloom Filter;
18. for (i = 64 down to 19)search dstIP%i;
19. if (V[h(/i)] = 1) return counters[i];
20. return BMP = fx(counters[i]) -> nexthop;}
```

先将路由器的默认下一跳存入 BMP（Best Matching Prefix）（第 2 行），然后提取目的 IP 地址的前 16bit 在 B-树上查找（3-14 行），具体查找过程如下，首先判断当前节点类型是 2-node 还是 3-node 类型，如果节点类型是 2-node，则数据项 *S* 赋值为 *L*（第 5 行）。然后赋值目的 IP 地址的前 16 位值为 *key* 并与数据项比较，在 *key* 与数据项都不相等的情况下，根据条件式决定转发至 *LNode*，*MNode* 或 *RNode* 查找。(6-10 行)。如果无匹配到的节点并且当前节点为叶子节点，说明路由表项中不存在匹配的前缀，转发数据包至默认下一跳。(第 12 行)。如果有匹配的节点，则返回节点中对应的向量 *V*（13-14 行）。最后在 Bloom Filter 中查找目的 IP 地址对应前缀长度前缀，如果找到，则返回 counters 数组值，并根据这个值对地址进行提取；如果没有找到，则转发至默认路由下一跳（16-20 行）。

图 7 为算法流程图，算法采用 B-树和 Bloom Filter 相结合的方式进行路由查找，因此图 7 共分为 2 大部分，左边部分为 B-树查找过程，右边部分为 Bloom Filter 查找过程。B-树部分查找过程如下：

a）截取当前需要查询的 IP 地址的前 16bit，并将它的值赋予变量 *key*，将节点变量 node 赋值为 B-树根节点；

b）第二步进入 B-树的遍历查找，首先判断 *key* 是否和节点 node 中的 *S* 相等，若等，则匹配成功，进入 Bloom Filter 查询 V_S 向量映射的路由表集合。若不等，则判断是否和节点 node 中的 *L* 相等，若等，则进入 Bloom Filter 查询 V_L 向量映射的路由表集合。

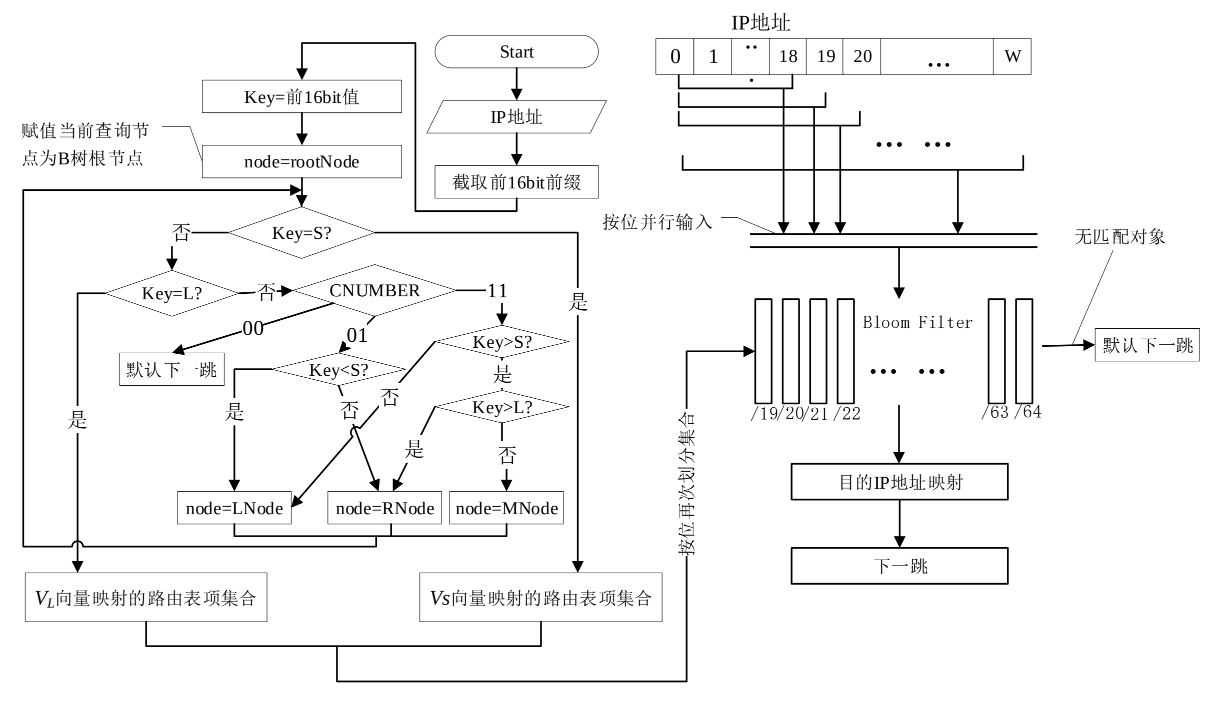


图 7　路由查找流程图

c）如果 *key* 和 *S* 与 *L* 都无法匹配，那么需要进入下一节点查询。首先根据 *CNUMBER* 取值判断 node 节点有几个孩子节点，若 *CNUMBER* 为 00，则表示其为叶子节点，此时匹配项为默认下一跳；若 *CNUMBER* 为 01，则表示该节点有两个子节点，当 *key* 值大于 S 时进入左节点 *LNode* 查询，小于 *S* 时，进入右节点 *RNode* 查询；若 *CNUMBER* 为 11，则表示该节点有三个子节点，当 *key* 小于 *S* 时进入左节点 *LNode* 查询，*key* 大于 *S* 且小于 *L* 时进入中节点 MNode 查询，*key* 大于 *L* 时进入右节点 *LNode* 查询。最后将查找到的 *LNode*，*RNode* 或 *MNode* 赋予 node 变量，进入第 b）步循环查找。

当目的地址 IP 的前 16 bit 在 B-树查找匹配成功后，将得到其值对应的向量 *V* 所映射的路由表项集合，在该集合里，按照前缀长度/19-/64 再次划分为 46 个小集合，每个集合对应一个 Bloom filter，然后进入 Bloom filter 查询阶段：

a）将目的 IP 地址按位从/19 至/64 并行输入到对应长度的 Bloom Filter 中进行查找；

b）如果 Bloom filter 验证没有匹配项，则进入默认下一跳。如果 Bloom filter 验证有匹配项，则根据 Bloom Filter 中的前缀

地址映射得到对应的下一跳。至此，查找流程结束。

2.3 路由更新及算法性能分析

2.3.1 路由的更新过程

BTBF 算法支持增量更新，在路由表发生变化时只需要局部更新数据结构而不需要重头重新构建整个数据结构。当需要添加一个新的路由表项时，需要分别执行B-树的插入操作并在Bloom Filter 中执行计数数组元素值增加操作。

a）B-树中的插入操作为：获取路由项的前 16bits 前缀值 *value* 和前缀长度 *n*，在B-树查找 *value* 并得到匹配节点，如果匹配节点的 *S* 或者 *L* 值有相同数值，则转发向量 *V*[*h*(/*n*)]元素索引的 Bloom Filter 进入第 b）步更新；如无此值，则在 B-树节点中插入该值，B-树（本章算法使用的是 2-3 树）会根据需要自动旋转以致平衡，最后，更新向量元素 *V*[*h*(/*n*)]，并转发至 *h*(/*n*)映射的 Bloom Filter 进入第（2）步更新。

b）Bloom filter 的插入操作为：第 a）步映射的 Bloom Filter 的 counters 计数数组探测到的元素值+1。

删除前缀信息的操作与更新操作类似，首先执行更新过程的第 a）步的 B-树查找操作，不过，第 b）步 counters 计数数组元素的值-1。此外，当 Bloom Filter 的 counters 数组值 C_n 为 0 时，表示该集合已无路由前缀，这时需要回溯至 B-树节点，向量元素 *V*[*h*(/*n*)]置 0，并释放地址映射。若向量 *V* 所有元素值均为 0，则需要回溯至 B-树，删除节点中相应数据项，最后执行自动平衡旋转操作。

2.3.2 算法性能分析

B-树的高度决定了查找效率，BTBF 算法采用的 2-3 树形结构，在最坏的情况下，也就是所有的节点都是 2-node 节点，查找效率为 lg N。最好的情况所有的节点都是 3-node 节点，查找效率为 log 3N 约等于 0.631lgN 距离来说，对于 1 百万个节点的 2-3 树，树的高度为 12-20 之间，对于 10 亿个节点的 2-3 树，树的高度为 18-30 之间。插入只需要修改与该节点关联的节点即可，不需要检查其他节点，所以效率与查找相同。

本文算法中 B-树仅负责查找前 16bit 前缀值，而当前骨干路由表的前 16bit 值只有几十个，可以形成较好的树形，因此时间复杂度约为 O(log3N)，而仅使用 B-树进行路由查找的 RBMRTs 算法查找效率为 O(logN)。本文算法每个节点至少可存储两个路由匹配项，最好情况下需要 N/3-N/2 个节点，RBMRTs 算法则至少需要 2N 个节点。

Bloom Filter 的插入/查询时间都是常数 O(*k*)，*k* 为 Hash 函数个数，使用 Bloom Filter 需要考虑到它的错误率（误判率），错误率 *p* 由以下公式确定：

$$p=\left(1-\left(1-\frac{1}{m}\right)^{kn}\right)^{k}\approx\left(1-e^{-\frac{kn}{m}}\right)^{k}$$

m 是位数组大小，*n* 是 Bloom Filter 查找的集合元素个数。上述公式表明随着 *m* 的增加，错误率会下降，同时随着插入元素个数 *n* 的增加，错误率又会上升，对于给定的错误率 *p*，最优 Hash 函数个数 *k* 由以下公式确定：

$$k=\frac{m}{n}ln2\approx 0.7\frac{m}{n}$$

最优的位数组大小 *m* 由以下公式确定：

$$m=-\frac{nlnp}{(ln2)^2}$$

根据以上公式，可以确定出 Bloom Filter 最优的位数组大小和 Hash 函数个数，从而大大降低它的错误率。假设将错误率控制在 0.001 左右，根据前文骨干路由器的数据统计可知，在数据最庞大的 2001::段，前缀长度集合根据数量可分为千级和百级，如/48 长度的为 2970 个，2620::段/32 长度的为 1362 个，为了方便分级定位千级的集合统一取 *n* 值为 2500，根据计算结果可知即使是大的集合，最优的位数组大小也仅为 *m*≈35Kb，最优 Hash 函数的个数 *k* 为 10.06，所以 *k*=10。与上述同理，百级集合取近似中位数 200，它的最优位数组大小和 Hash 函数个数分别为 2.8Kb 和 10 个。相比 SARANG D[6]和 J. H. Mun[10]提出的 Bloom Filter 类路由查找算法，本文在两次划分集合后使集合元素个数和 Hash 函数的数量都得到大幅度缩减。同类型的 Bloom Filter 需要查找整个路由表项，Hash 函数个数在 13 个以上，不仅增加了误报率，还增加了查找时间，降低查找效率。

3 实验验证

实验阶段收集了 IPv6 骨干路由器自治域 AS6447、AS131072 和 AS3356 的路由前缀，使用 C++语言在 Visual Studio 平台上编写了测试代码。首先统计了三个自治域的路由表项，根据统计结果和上文中 Bloom Filter 最优数组大小 *m* 和 Hash 函数个数 *k* 值的公式，将经过 B 树分类后的路由表项集合进行了分类处理。分类结果与上文相同，确定了两个级别：千和百级，*m* 和 *k* 分别为 35Kb、10 和 2.8Kb、10。对应的错误率为 0.001。实验中为了符合实际情况，均使用各自治域的活跃路由项。

表 1　三个自治域中前缀统计

路由表项数	AS6447	AS3356	AS13172
前 16bit 为 2001	2899	3619	5440
前 16bit 为 2600	1167	1609	2287
路由表项总数目	48348	56444	85899

在实现 Bloom Filter 时多采用简单、性能较优的折叠异或哈希函数。生成了 10 万个数据包进行测试，为了验证数据包的大部分都有匹配条目，85%的数据包是从路由表中选择的路由条目，另外 15%的验证包是完全随机生成的，这样验证包可以匹配到路由条目，也可以匹配到默认下一跳。为了验证 Bloom Filter 的错误率稳定性，分别对以上三个自治域的 2001::段和 2600::段做了 Bloom Filter 错误次数的统计。统计结果如图 8 所示。从图 8 可以看出实验结果基本与算法设计相符，错误率保持在 0.001 左右。在实际应用中，还可以根据每一个集合的大小定制最优的 Bloom Filter，从而使错误率更加符合预设。

算法内存消耗主要用于 B-树、Bloom Filter 的位数组和路

由表项的加载。在内存消耗的实验中，使用 AS131072 的 BGP 路由表，BTBF 算法的实现占用内存保持在 2700 KB，其中绝大多数用于对路由表的加载。表 2 给出了内存消耗比较的结果。

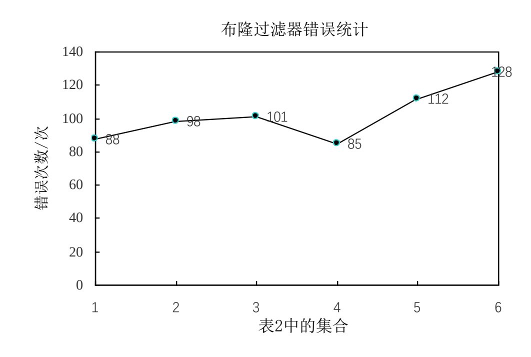


图 8 Bloom Filter 错误次数统计

表 2 真实路由表测试内存占用比较结果

查找算法	表项数	内存
本文算法	85899	3.16M
基于 Hash 和 CAM 的 IPv6 路由算法	65535	15.33M
基于 Bloom 滤波器的快速路由查找	25000	4M
BSRPS	78430	4.8M

理论上 BTBF 算法时间复杂度最多为 O(5)。查找性能部分的实验，继续使用上文中的 10 万个数据包进行测试，并与 RBMRTs 算法，BSRPS 算法和 Bloom 滤波器路由查找算法进行了比较。RBMRTs 算法的查找树内部节点中存储了许多重复的端点，当路由表项增大时，重复的端点也会成倍增加，导致不能输出稳定的性能。

BSRPS 算法是根据前缀值区间所做的集合树进行查找，其性能表现稳定，但是由于 IPv6 的海量地址空间，集合树的深度不能控制到一个较低的级别，所以性能并不是十分良好。

基于 Bloom 滤波器的算法采用了首字节索引的方式排除了一部分不在路由表中匹配的 IP 地址，降低了对 Bloom 过滤器的探测次数，不过，当被查询地址的首字节对应的索引数量较多时，首字节索引方式较并行查询的优势会缩小，滤波器数量的减少导致错误率和查找时间增大。图 9 中给出了这三种算法与 BTBF 算法在同一环境下的性能对比结果。

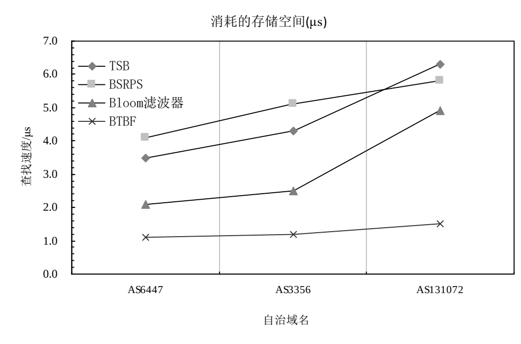


图 9 用生成的路由表测试的查找性能比较结果

从图中可以看到，随着表项的增加，算法的查找速度一直趋于稳定，并且保持在较高的性能上。这是由于 BTBF 算法对前缀进行了 B-树和前缀长度的分类，这两种方式都能适应数据量的巨大变化，Bloom 过滤器的查找集合一直保持在稳定的大小。

4 结束语

分析了 IPv6 地址结构，分配策略和 IPv6 骨干路由表的特征，将 B-树和 Bloom 过滤器相结合，提出了一种基于 B-树和 Bloom 过滤器的 BTBF 算法。与已有算法相比，BTBF 算法查找速度快，占用内存小，有较好的扩展性，并且支持增量更新。实验结果表明，BTBF 算法在保持优良的查找性能外还能适应路由表大小的改变，始终保持稳定状态。这得益于对 B-树查找深度低这一优良特点的充分利用，促使整个算法的正确性和可靠性，实验结果也验证了该方法的有效性。

参考文献：

[1] 李振强，郑东去，马严. TSB：一种多阶段 IPv6 路由表查找算法 [J]. 电子学报, 2007, 35 (10): 1859-1864. (Li Zhenqiang, Zheng Dongqu, Ma Yan. TSB: a multi-stage algorithm for IPv6 routing table lookup [J]. Acta Electronica Sinica, 2007, 35 (10): 1859-1864.)

[2] Hsiao Yimao, Chu Yuansun, Lee Jengfarn, *et al.* A high-throughput and high-capacity IPv6 routing lookup system [J]. Computer Networks the International Journal of Computer & Telecommunications Networking, 2013, 57 (3): 782-794.

[3] Bando M, Chao H J. Flashtrie: hash-based prefix-compressed trie for IP route lookup beyond 100Gbps [C]// Proc of IEEE INFOCOM. 2010: 1-9.

[4] Zhong Pingfeng. An IPv6 address lookup algorithm based on recursive balanced multi-way range trees with efficient search and update [C]// Proc of International Conference on Computer Science and Service System. 2011: 2059-2063.

[5] 崔宇，田志宏，张宏莉，等. 基于前缀区间集合的 IPv6 路由查找算法 [J]. 通信学报, 2013, 34 (06): 29-37, 48. (Cui Yu, Tian Zhihong, Zhang Hongli, etal. Bianary search on range of IPv6 prefix sets [J]. Journal on Communications, 2013, 34 (06): 29-37+48.)

[6] Dharmapurikar S. Longest prefix matching using bloom filters [J]. IEEE//ACM Transactions on Networking, 2006, 14 (2): 397-409.

[7] 李鲲鹏，兰巨龙. 基于TCAM 的高效浮动关键词匹配算法 [J]. 计算机工程, 2012, 38 (4): 269-274. (Li Kunpeng, Lan Julong. Efficient unfixed keywords matching algorithm based on TCAM [J]. Computer Engineering, 2012, 38 (4): 269-274.)

[8] 于明，王振安，王东菊. 基于 Bloom 滤波器的快速路由查找方法 [J]. 哈尔滨工程大学学报, 2014, 35 (10): 1247-1252. (Yu Ming, Wang Zhen’an, Wang Dongju. A fast method for IP lookups based on Bloom filters [J]. Journal of Harbin Engineering University, 2014, 35 (10): 1247-1252.)

[9] http://bgp. potaroo. net [EB/OL]. 2017.

[10] Ju H M, Lim H. New approach for efficient IP address lookup using a Bloom filter in trie-based algorithms [J]. IEEE Trans on Computers 2016, 65 (5): 1558-1565.